INTRODUCTION

Today's School: Wired for Tomorrow

The Internet has enormous potential to create an environment that fosters the growth of lifelong learners. Internet resources can be important of engaged learning, where students can structure their learning within a more project-oriented curriculum, addressing real-life problems and using the most current news, data and information. The Internet can allow us to go beyond drill and practice or rote memorization to provide students with the authentic opportunities to develop complex thinking and problem solving skills. With the Internet, the classroom can expand to include interaction with students; with experts in the academia, museums, business, government; and with the community. If students have a lengthy exchange with an expert, they can begin to understand how, for instance, mathematicians or scientists think, what kinds of real-life problems they must solve, and how a simple tool such as email helps them to collaborate.

When we talk about engaged learning, students: (1) Take an active role in meaningful tasks; (2) Are responsible for their learning; (3) Demonstrate their understanding; (4) Explore a variety of resources; (5) Strive for deep understanding; and (6) Work toward high standards of achievement. The Internet presents a new and exciting opportunity for students and teachers to contribute to the larger body of information and to the learning of others through online communication, collaboration, and publishing.

The Internet and the Changing Role of the Teacher

Internet resources can help build the skills students need to succeed in the Information Age. More and more, today's workplace requires the ability to gather, evaluate, synthesize, and apply information, in addition to considering several possible solutions rather than seeking one right answer. Graduates will need teamwork experience and collaborative problem-solving skills. Teachers can help students construct questions, decide how best to answer them, conduct research and collaborate with fellow learners in and beyond the classroom. Internet tools and resources, along with desktop productivity software, can help teachers teach students to store, retrieve, manipulate, analyze, and apply information. While there may be less emphasis on information delivery on the part of the teacher, there are more opportunities to teach and model higher level of critical thinking skills.

Concept: Student-Centered Classroom

A less teacher-centered classroom also allows students to step into a teaching role. Both teacher and students can learn from the technically savvy student, and everyone benefits from this reversal of role. With basic skills and procedures in place, the teacher in the classroom can begin to act more as a facilitator, or coach. The classroom can be become less teacher-centered, with students taking more responsibility for their own learning. Lecture styles information delivery still have its place, but the teacher can be more of a model or mentor, one who guides, suggests, points and helps to connect students with experts and other resources.

Focusing on the Objectives of a Computer Science Course for a School Environment

To a student, a computer is simply a tool – albeit and interesting one, that one may use to produce a lot of the so-called "hi-tech" stuff seen in television commercials, movies, video games and some elaborate printed materials. To ask a student to use the computer may elicit a host of reactions, ranging from fear to eagerness, but alas, whether they want it or not, the use of computers is a necessity in this age, simply because it is becoming an integral part of the different activities of today's society.

When imparting knowledge, the general method is through a widening spiral curve where the basic concepts of what and how are initially done, expounding on these concepts and tackling them on a deeper and more profound matter as the student grows older and more mature, and is consequently able to integrate concepts on a grander scale. Computer science and technology are essentially of the same nature. There are some concepts that are easily understood at a certain age, and there are some that can only be learned at a later stage. There are also students that can easily grasp certain concepts while some take a longer and more arduous time to understand things.

The high school level is just the right age for students to learn about things. It is also the stage in child's life when he already becoming aware of the responsibility and is into the age of looking ahead for consequences before doing a certain act, hence the concepts of computer viruses, the do's and dont's in technology processes, and the over-all consciousness of the environment may be taught. It is also important for these kids to get a first hand experience of the applications of these concepts – hence the need for activities that will make them apply the knowledge they learn, and make them appreciate and get a clearer understanding of the underlying concepts.

Moreover, more and more universities are offering computer science subjects specifically computer programming as part of engineering courses' basic requirements. The subject is taught either by direct computer programming class usually using Java or by Program Logic Formulation course.

RATIONALE

The Rationale in Teaching Computer Programming

When teaching computer programming to students, it is important to remember that these children are not computer science majors, and that their understanding of the subject vary extremely. Strangely enough there is no exact definition of computer literacy, so that the issue of programming as part of a high school computer science curriculum has been raised time and again.

I would like to believe that when one has a sufficient knowledge and understanding about programming logic, he is in a better position to identify what type of software he needs for a particular application because he has a healthier appreciation of the powers of computing. Therefore, to join the fray, computer literacy does not only mean knowing how to operate the computer and navigate through certain software. To be considered computer literate, one must also understand the capabilities and limitations of computers, and be able to identify the aspects of computer technology that mat be used as a tool for whatever project is on hand.

OBJECTIVE

Course Objectives

Because the objective of teaching computer programming to high school students is to promote computer literacy, the extent of programming must be confined to certain areas that will be considered as a necessary for gaining a healthy appreciation of computing power. The following are the specific objectives of the course:

1. To encourage and train the students to develop their logical thinking abilities by exposing them to problem-solving activities using computers;
2. To help the students efficiently and effectively organize data into logical structures usable for information processing; and
3. To equip the students with the necessary tools foe producing well-organized, systematic and methodical solutions to problems involving complex data manipulations.

The emphasis in teaching programming to students is on the program logic formulation, and not on programming languages. The language is only necessary to illustrate the concepts in a more concrete setting. In choosing a language, however, one must take into consideration a language that is structured and encourages modularity.

What is the amount of knowledge that a student must have to be assessed as ready for learning about computer programming?

Preparation is a key factor in making students understand programming. Before anything else, the student must be able to think as the computer does, that is, he must have a clear understanding of how the computer processes data in terms of memory management; classification and representation of data; treatment of literal and named constants, and variables; sequencing and hierarchy of operations; and the formation of conditions into logical expressions.

To achieve an understanding of memory management, the organization of computer systems must be made clear. Specifically, the student must know about how data are stored in the main memory, retrieved and transferred to the CPU registers for processing, and the results brought back to the same or another address in the main memory. This concept lays down the foundations for understanding about variables and how they may change or retain their values after a sequence of operations.

Although there is no need for the student to understand about how the computer recognizes different types of data, he nevertheless needs to know how each one is different from another from the user's point of view. This includes the identification of the range of values represented, and the domain of the symbols used.

The differentiation if literal constants from named constants, and constants from variables is a topic that is often simply glossed over but actually, a good understanding of this aspect leads to easier visualization of the concept of assignment statements. As for the use of named constants, pointing out its advantages will greatly help the students in later activities in programming.

Operations and the rules on sequencing and hierarchy must also be fully understood since solutions to problems will more often than not make use of operations in order to manipulate data. The student must learn how to correctly process the information the computer way, so that when the time comes for him to study about logic formulation, he can concentrate on the logical structures of the program, and not on the structure of expressions. Due emphasis must be given on word problems being converted into numerical or logical expressions, most especially on the latter. Conditions are often in a lot of programming constructs, and it is crucial that the student be properly trained in recognizing the conditions in a situation and transforming them into the proper format of relations and eventually, into logical expressions.

FORMAT

Format of Classroom

The following are the usual activities in learning that are included in teaching computer programming:

- The usual lecture sessions and discussions
- Presentations
- Written Exams (Quizzes, Midterm Exams, Final Exams)
- Assignments
- Exercises (Games, Board Works, Kinesthesis)
- Problem Sets

There is no exact formula that may be followed for the implementation of these activities. The format or sequencing of activities largely depends on the topic under discussion and the responses of the students to the activities. Most often, the first three activities are jumbled up on a week or by topic scenario:

Lecture → Written Exams → Exercises
Presentation → Exercise → Discussion → Written Exams
Exercise → Lecture/ Discussion → Written Exercise

Lecture sessions may take the form of traditional teaching using the board, but an actual demonstration of how a command or statement works enhances understanding a lot. It is still important though to first explain the concept behind a certain command or structure, and then throw to the class some hypothetical variations of the command for their interpretation, before demonstration is done. In this way, the students are encouraged to evaluate their own understanding of the basic theories taught without depending on the computer to prove whether they are right or wrong. Since it is the student's logical thought process that is being developed, and not the computer's, then it's the students whose performance must be evaluated.

Discussions, in the context of this paper, refer to post-activity lectures where the activity's expected results are presented and the different results of the students' performance of the activity. These sessions are more for giving out some helpful tips, and for emphasizing certain points in the activity.

Written exams must reflect the lesson/s recently discussed, and may include some other concepts discussed and worked on previously as a check-up on the students' retention of lessons. Ideally, the purpose of written exams is to make the students apply whatever knowledge they have gained onto the problem. Purposely, solutions to problems may not be tested immediately on the machine. This is to encourage the students to carefully think of the solution and to detect the errors by tracing their own programs. Memorization is not encouraged in programming; understanding is, hence written exercises are recommended as an open-notes session. Problems given should be easy enough to be done by the student within the allotted time for class, but should a topic be too complicated to have simple application, the exercise could concentrate on the creation of a program segment only. Tracing could also be done as a written exercise to train the student in understanding the logic behind a program code. Written exercises are recommended as an individual activity.

Quizzes are interspersed into the sequence of lectures and activities (usually after a lecture) for testing for the more objective and concise aspects like syntax, characteristics of some basic concepts, terminologies, and the like. The inclusion of logic-formulation in a quiz (where the student is asked to program or construct a certain part of the program) is not really that advisable because it takes too much time for the student to think of the solution for the problem. Consequently, in a one-hour period, half the time is taken up by the quiz portion.

Construction of written examinations must be done with a lot of thought so that analysis is emphasized more than the need to memorize syntax and other objective topics. Asking a student to create a program during an exam might be asking a bit too much of them, since an exam is usually only for an hour, and to ask a student to create a program in an hour is an anti-thesis to the goal of making the students think carefully and thoroughly of the solution. If any programming is to be done during an exam, it should be in the form of a program segments only, or else the number of expected lines of code should be limited to about 30 lines.

From the point of view of the teacher, practical exams are not very practical, unless there are enough computers to have all students working on the exam all at once. Technically speaking though, the true test for knowing whether the students has really know how to program or not is through this method. Then again comes the issue of what we expect the students to gain from the course. Since programming is not the real objective of the course, but the logic behind programming, practical exams are not highly recommended.

In same light as the practical exams, projects are not very high on the list of activities for a course in programming if we are to focus on our objectives. More often than not, there is also the tendency of some students to depend on others for the completion of a project, or to let others do their work altogether.

Grading of Programs

How may programs be graded so as to attain a high objectivity rate? In grading programs, we must first establish the emphasis in the exercise. For sure, there are two main aspects of the program that must be considered: the syntax and the logical structure of the program. Depending on the nature of the exercise, the following are other points that may be given weight:

- Creation of variables
  - relevance of name to purpose
  - correctness/ appropriateness of data types
- Input to the program (is it sufficient to do the required data processing?)
- Output of the program
- Correct initialization of variables

- Correct sequence of operations on the data
- Correctness/ appropriateness of logical structures used
- Proper grouping of statements into a compound structure
- Readability of the program
- Formatting of input/ output for user-interface

PROGRAM LOGIC FORMULATION

With the advent of history we also saw the advancements in the computing methods. The Chinese developed the Abacus. In 17th Europe, various calculating devices were also developed. Blaise Pascal invented a mechanical calculator in 1642 and Gottfried Leibniz came up with the multiplier in 1673. The slide rule invented by William Oughtred's was used by engineers and scientists for three centuries.

The Industrial Revolution also triggered the invention of other calculating devices some of which are programmable. These include Joseph Jacquard used punched cards for the loom in order to do pattern weaving. Charles Babbage invented the Difference Engine and the Analytical Engine. At the close of the 19th century, Herman Hollerith invented the Census Machine used in the 1890 US census.

The 20th century marked the beginning of the computer age which is now known as the computer age. John Atanasoff built the ABC- Atanasoff-Berry Computer. During WWII, Konrad Zuse built a wartime computer, Britain came up with COLOSSUS, the first single purpose electronic computer, and Harvard professor Howard Aiken made the first general-purpose electro-mechanical computer operational. In 1946, ENIAC, the first general-purpose electronic computer.

In 1945, John von Neumann developed the concept of storing the program in the computer's memory, the *stored program concept*. He also came up with the six components of his theoretical computer which are:

1. arithmetic unit – for basic computation
2. logic unit – where decisions and comparisons can be performed
3. input device – accepts coded instructions and numeric data
4. memory unit – for storing instructions and data
5. control unit – for interpreting the coded instructions and controlling the flow of data
6. output unit – communicates the results

Generations of Computers

First-Generation Computers (1951-1958)
- vacuum tube technology
- punched card or magnetic tape
- machine language
- magnetic core
- e.g. UNIVAC I, IBM 650

Second-Generation Computers (1959-1964)
- transistor
- solid-state technology
- punched card or magnetic tape
- assembly language and some high-level languages
- magnetic core
- e.g. IBM 1401, GE 235

Third-Generation Computers (1965-early 1970s)
- IC (integrated chip) technology
- silicon chips
- large-scale integration (LSI)
- punched card, magnetic tapes and disks
- magnetic core, semiconductor memory
- e.g. IBM system/360, DEC PDP 8

Fourth-Generation Computers (early 1970s-1980s)
- VLSI (very large-scale integration)
- microprocessor chip
- magnetic and floppy disks, etc
- high-level languages
- user-friendly software
- semiconductor memory
- e.g. IBM system/370, Apple Mac, SPARC II workstation

Fifth Generation (1990s)
- voice activated/ response system
- accepts graphic images as input

- translates foreign languages
- able to learn from own experience and program itself

Uses of Computers

- Numerical computations
- Data storage and retrieval
- Word processing
- Computer graphic and animation
- System simulation
- Artificial intelligence
- Communications and networking
- Desktop publishing
- Databases
- Management information system
- Spreadsheets

Components of a Computer System

1) **Hardware** – refers to the physical component of the computer system.

   CPU – Central Processing Unit

   a) control unit- directs the actions of the system by carrying out the instructions and establishing their sequence according to the program.
   b) ALU- performs basic arithmetic and logical operations (comparisons); contains registers which can hold single data item.

   Memory- consists of addressed locations that can contain instructions or data items; data can written to or read from these memory locations.
   Two types: Primary and Secondary

   I/O Devices- for communication and data transfer (keyboard, monitor, mouse, printer, network, peripherals etc.)

2) **Software** – refers to the program / instructions.

   System Software- set of programs that belong to the configuration of the computer.
   Application Software- written for obtaining particular solutions to problems

3) **Users**- programmers, simple users, etc.

Information Flow in Computers (4 elements of computer process)

Figure 1:

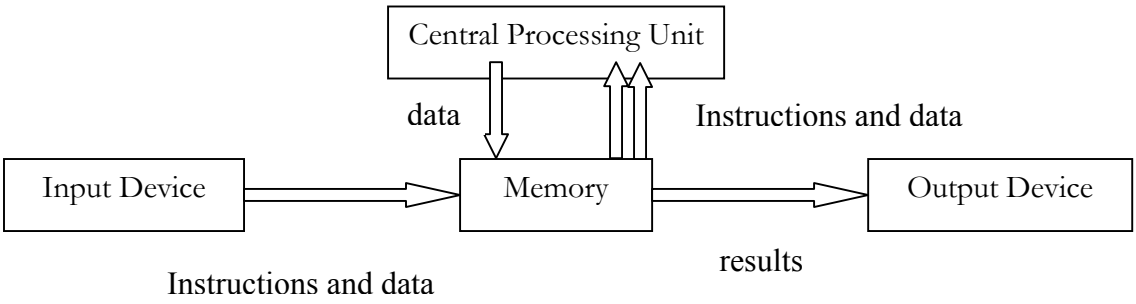
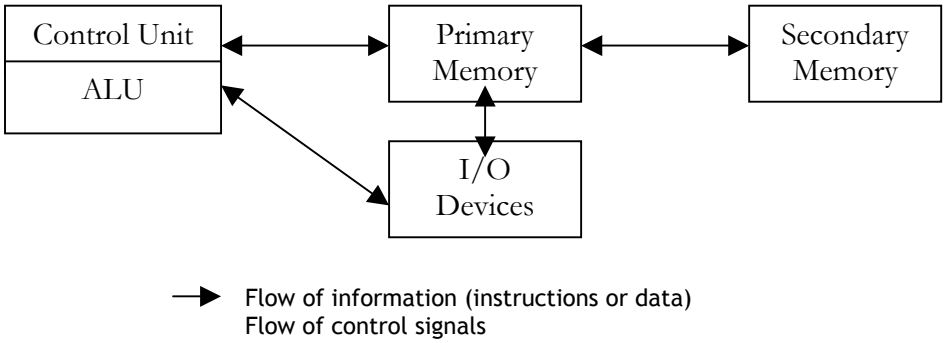


Figure 2:



Introduction to Programming

A. Programs and Programming Languages

Program- list of computer instructions required to arrive at the results.

Programming language- a system of notations and rules used for writing programs

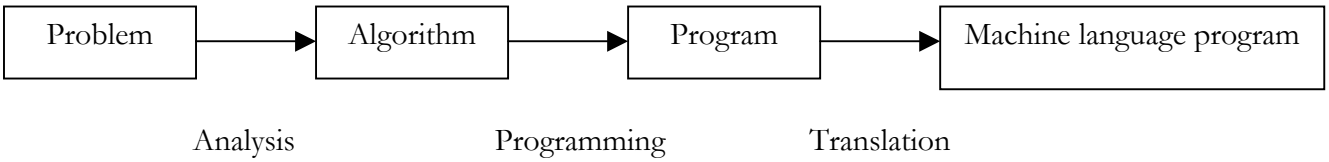
Algorithm

- precisely expressed procedure for obtaining the problem solution
- finite set of instructions that specify a sequence of operations to be carried out by the computer
- independent of both the languages in which they are expressed and the computer which executes them
- Al-khowarizmi – 9th century Arab mathematician and astronomer

Algorithm may be expressed either as a:

1. **Flowchart** – a system of symbols for expressing algorithms; indicates flow of control/ sequence of operations.
2. **Pseudocode** – textual representation of an algorithm, close to natural language, becomes part of the program documentation.

Algorithm : Human as a Program : Computer



B. Seven Basic Elements of Programming

1. **Data:** constants, variables
   e.g. Donald, 244234, `hello`, name, age

2. **Input:** reading of values from input devices (keyboard, I/O port disk drives)
   e.g. READ (name)

3. **Output:** writing of information to any output device (screen, disk drive, I/O port)
   e.g. WRITE (name)

4. **Operations:** comparing values, assigning values, combining values
   e.g. a<b a¤10 a+b

5. **Conditions/Selections**
   e.g. IF THEN ELSE, CASE/SWITCHES

6. **Loops/Iterations**
   e.g. WHILE DO, REPEAT-UNTIL, FOR DO

7. **Subroutines/Modules**
   e.g. functions, procedures

C. Steps in Programming Development Cycle

1. **Problem Definition**
   - The particular problem to be solved must be clearly defined.
   - Assumptions regarding inputs to the program and the desired output must be made

2. **Problem Analysis**
   - Determine the most effective and efficient approach to solving the problem.
   - Breakdown of the problem into tasks.
   - Incorporation of existing solutions.
   - Data pertaining to the problem must be gathered and analyzed.

3. **Design of Data Structure**
   Determine the data composition to be used in the program:
   - desired out
   - input data
   - general processing procedures required to convert input data into output results

4. **Design of the Algorithm**
   The most crucial stage of problem solving with computers

The solution to the problem is described in an algorithmic notation
Specifications is broken down into:

- input/output
- calculations
- logic comparisons
- data storage/retrieval operations

5. **Coding**
   Algorithm is written in a chosen programming language

6. **Program Testing and Implementation**
   Testing of the program in order to check for errors and its response to all possible data inputs.

   Types of Errors:
   Syntax: error in notation or grammar; occurs during compilation
   Semantic: error in logic; can be detected during runtime

7. **Program Documentation**
   - Description of the program operations, data structures, and I/O specifications.
   - A reference manual and a user's manual are developed alongside with the software/program. These manuals are used for instructing the user how to install and use the program, and to help the developer of the program in the future modifications of the program.

8. **Program Use and Modifications**
   Actual use of the program, and modifications are performed for the following purposes:
   - correcting errors not detected during testing phase
   - adapting into a new environment
   - enhancement, addition of new features

Properties of Algorithms

1. Finiteness
   - the algorithm execution must be completed after a finite number of times.
   - execution time: indicated by the number of times a step or steps are repeated and not on the program length.
   - the number of steps performed depends on the input data
   - number of operations is not equal to the number of steps in the algorithm
2. Absence of ambiguity
3. Sequence definition
4. I/O definition
5. Effectiveness
6. Scope definition

LANGUAGE ELEMENTS

A. Labels/Identifiers

- Used as names for variables and other items in the program.
- The format depends on the programming language.

Rules for forming identifiers

1. Can be any string composed of letters, digits and underscore (_)
   No special characters.
2. Should begin with either a letter or an underscore.
3. Should not be more than 128 characters
4. Must be descriptive

e.g. _I_AM_IDENTIFIER, ABC123

Identifiers can be case-sensitive or case-insensitive

Case sensitive- lower case (small letter) is not the same as upper case.
e.g. x is not the same as X

Case insensitive- lower case (small letter) is the same as the upper case.
e.g. num is the same as NUM

B. Variables

- Data items whose values can change during program execution and for locations in memory are reserved.
- Variables can be categorized (or "typed") by the kind of data it can hold (e.g. integer, character, string). They must hold data that are of the same type, otherwise a mismatch error will occur.

Variable name- label/identifier assigned to a variable

Content value- piece of data inside a variable. The value of a variable may be assigned specifically or be a result of operations.
e.g. A = 6

A
6

C. Constants

- Data whose values never change.

Two kinds of constants:

Literal- refers to the actual value itself. e.g. 60, `z`

Named- uses a name or alias to represent an actual or literal value

Constant types
- Integer constants: whole numbers, positive/negative; 87, -766, 244
- Real constants: decimal numbers including integers, positive, negative; 53.535, 0.093, 88
- Floating point constants: positive/negative numbers in exponential form or scientific notation:
  5E2 = 500 = 5
  X 10^2: 5 – mantissa, 2 – exponent
  -1.82E2 = -182 = -1.82 X 10^2
  2.43E-4 = 0.000243 X 10^{-4}
- Character constants: any ASCII character: 'a' – 'z' – 'A' – 'Z', '0' – '9', '&', etc

D. Command verbs

- Used to give instructions to the computer.
  e.g. Input and Output commands such as Get and Show etc.

E. Operator

- The computer processes data by performing operations. Operators are the symbols used to represent an operation to be performed.
- The most common kind of operations are arithmetic/numerical, relational and logical.

F. Expressions

- Literal constant
- Named constant
- Variable
- Combination of operators and operands

G. Separators

- Used to separate the data items and statements in the program
  e.g. comma (,): used in declarations, input/output statements
  semicolons: used to end the statement (PASCAL, C)

H. Comments

- Written to describe what a statement does or for what a particular declaration is:
- Symbols used to enclose or begin comments
  (* *), /* /, //
  e.g. (* This is a comment *), // this is also a comment

DATA MANIPULATION IN PROGRAMMING

A. Data Representation

1. ***Simple Data Types*** for single data items
   a. Numeric
      Integer, Real, Float
   b. Character
      Format: Char (<length>)

2. ***Structured Data Types*** for groups of (related) data
   - Array

- List
- Queue

B. Operators and Expressions

1. Numerical

Perform mathematical operations:

- + addition
- − subtraction
- *multiplication
- / (div) division
- mod modulus arithmetic (remainder)
- ^ or ** exponentiation

Rules to follow in performing numerical operations

a. Follow order of priority or precedence
   Exponentiation
   Multiplication/Division
   Modulus
   Addition/Subtraction
   e.g. 3*5+10/2

b. Evaluation is done from left to right for operators with the same priority
c. Operations inside parentheses are evaluated first

2. Relational

Table of Comparison

Operator	Meaning	Example (RELATIONS)
=	Equal to	A = B
<>	Not equal to	A <> B
<	Less than	A < B
>	Greater than	A > B
<=	Less than or equal to	A <= B
>=	Greater than or equal to	A >= B

3. Logical

- Used to evaluate relations or combinations or relations

The most common logical operators are AND, OR, NOT

A	B	A and B	A or B	not A
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

4. Character String

- Concatenation: +
  e.g. 'hello' + 'world'

Rules for evaluating expressions (combination of operators)

1. binary operators: not
2. multiplying operators: *, /, mod, and
3. adding operators: +, - or
4. relational operators: =, <>, <, >, <=, >=

C. Statements

1. Input Verbs

Input command verb: signifies that a value for a particular data item (variable) is read either from the keyboard or loaded from a secondary storage.
e.g. Accept, Input, Get, Read, Enter, Load, Obtain
Format: <Input command verb> (<variable name/s>)

Output command verb: signifies that a value of a certain data item is to be given out as a result either as a screen output or as a printout.
e.g. Say, Write, Output, Display, Show, Type, Print
Format: <Output command verb> (<variable name/s> or <constant values>

Data Type

1. int- byte ,word, long int
2. Real Float
3. Char (256)
4. String- Array of

2. Assignments

Used to assign or give values to variables
Format: < Receiving variable> <assignment operator> <expression>

Receiving variable: can be any valid variable declared in the program
Assignment operators: :=, =, <-
Expression: any valid expression that returns a value.

3. **Declarations**

Denote data items (whether variables or constants to be used in the program.
Format:

a. Variable Declaration: <Variable name> = <Data type>
b. Constant Declaration: <Constant name> = <constant value>

PARTS OF A PROGRAM

Program header (optional)
Declaration part • Where variables, constants and • Non-executable
Program body • Where statements the instruct the computer are written • Executable

THREE BASIC FORMS OF ALGORITHM STRUCTURES (Control Structures)

A. SEQUENCE

An algorithm is said to be a sequence of steps when

a. the steps are executed one at a time
b. each step is executed exactly once; none is repeated and none omitted
c. the order in which the steps are executed is the same as that in which they are written
d. termination of the last step implies termination of the algorithm

Disadvantages of sequenced algorithm

a. inflexible: provides no alternatives
b. primitive in structure
e.g. Get (Num1, Num2)
Sum= Num1 + Num2
Diff= Num1 – Num2
Print (Sum)
Print (Diff)

B. PROGRAM BLOCK/ COMPOUND STRUCTURE/ COMPOUND STATEMENT

- A group of steps or statements that aim to achieve a single result
- Enclosed by a pair of delimiters such as BEGIN and END
- The statement(s) may be any statement such as assignment, input, output, decision and loop statements.

Syntax:

```
BEGIN
        Statement 1
        Statement 2
        …
        Statement  N
END
```

Example:
```
BEGIN
Get (Num1, Num2)
Sum = Num1 + Num2
Diff = Num1 – Num2
Print (Sum)
Print (Diff)
```

C. DECISION

Allows for two alternative actions or steps to be taken. For a particular step to be taken, a condition must be satisfied, otherwise the other step is taken.

If Then Else

Syntax:
```
If condition
then statement 1
else statement 2
```

The statement may be single or compound.

e.g.
```
if a number Num is zero
then don't use it as a divisor
else use it as a divisor
```

Conditions are expressed using logical expressions. They may be simple like a relation or compound (combination of relations and logical variables connected by logical operands)

Examples of logical expressions:

Is the number X equal to Zero?	X=0
Is the number Y greater than 1?	Y>1
Is the number Z between 0 and 100 exclusive?	(Z>0) and (Z<100)
Is the number X between 0 and 100 inclusive?	(X>=0) and (X<=100)
Is Y outside the range 0 to 100?	(Y<0) or (Y>100)
Is the sum of A and B less than C?	(A+B)<C

Nested Decisions

Allow for several alternatives to be taken. Use nested if – then – else

(1)
```
If condition1 then
   If condition then
      Statement 1
    Else
      Statement 2
Else
   If condition then
      Statement 3
   Else
Statement 4
```

(2)
```
If condition 1 then
     Statement 1
Else if condition 2 then
    Statement 2
Else if condition 3 then
    Statement 3
```

e.g.
```
if Month= 1 then
        PRINT ('JANUARY')
Else if Month= 2
        PRINT ('FEBRUARY')
Else if Month= 3
        PRINT ('MARCH')
Else Month= 12
        PRINT ('DECEMBER')
Endif
```

Selection Statements

The above nested if statements can be written using selection statements. A selection statement allows selection of a particular action from a set/list of alternatives

Syntax:
```
SELECT (<Expression>)
        Case constant 1: Statement 1
        Case constant 2: Statement 2
        …
        Case constant N: Statement N
ENDCASE
```

The expression (selector) can be any valid expression that must result to a constant value.
The result of the expression is compared with the choices (constant) and the statement for the matching constant is executed. The statement(s) may be any single statement such as assignment, input, output, decision and loop statements or a compound statement.

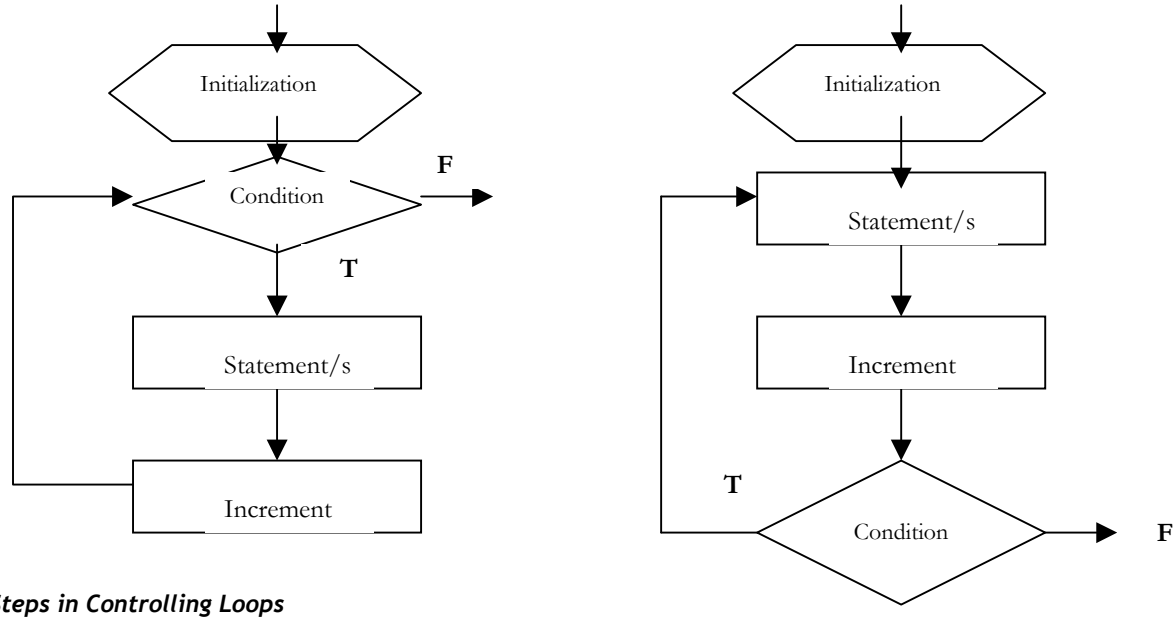
D. ITERATION

An algorithm structure that allows a statement or a set of statements to be executed *(to iterate)* for a certain number of times while some condition is TRUE or until some condition is TRUE.

E. LOOP

The instruction, tool or mechanism used for iterations.

Structures of Loops



Steps in Controlling Loops

1. Initialization

This process is usually done before the execution of the loop. The value of the *counter variable* is iniatially set equal to zero or one or to any appropriate value. The counter is the variable used to keep track of the number of iterations/repetitions of the loop.

2. Test for limit conditions

Before loop execution ends, a *loop terminating condition* (LTC) must be satisfied. This LTC is usually found at the beginning or at the end of the loop. The LTC is written as a Boolean or logical expression.

3. Incrementations

The value of the counter is usually increased by 1 or any appropriate value. This process is done inside the loop.

Types of Loops

1. WHILE-DO / DO WHILE

Syntax:
```
Initialization
WHILE condition DO
        Body
```

The body can be a single statement or a compound statement. When the while-do statement is executed, the condition is evaluated first. If it evaluates to TRUE then the Body is executed. Otherwise (condition evaluates to FALSE), the program goes out of the loop. This testing for the condition is done repeatedly until the condition evaluates to FALSE.

Example. Display the numbers 1 to 5

```
Counter=1
WHILE Counter <= 5 DO
BEGIN
    PRINT Counter
    Counter= Counter + 1
END
```

2. **REPEAT-UNTIL**

Syntax:
```
Initialization
REPEAT
Statement1
        Statement2
        …
        StatementN
UNTIL condition
```

The statements are first executed sequentially and then the condition is evaluated. If it's TRUE then the execution goes out of the loop, if not then the statements are executed again.

Example: Display the numbers 1 to 5

```
Counter=1
REPEAT
    PRINT Counter
    Counter= Counter+1
UNTIL Counter > 5
```

3. **FOR-DO**

Syntax: FOR Counter = INITIAL EXPRESSION TO FINAL EXPRESSION DO
Body

The INITIAL EXPRESSION results to value that is assigned to counter (Initialization). The Body can be single or a compound statement and is executed several times until the Counter variable is already equal to the value of the FINAL EXPRESSION. The value of the counter must not be modified inside the BODY.

To know the number of times the loop is executed, use the following formula:
No. of Iterations = (FINAL EXPRESSION – INITIAL EXPRESSION) + 1

For decreasing values:
Syntax: FOR Counter = INITIAL EXPRESSION DOWNTO FINAL EXPRESSION DO Body

Example: Display the numbers 1 to 5

```
FOR Counter= 1 to 5 do
        PRINT Counter
```

Example: Display the numbers 5 to 1

```
FOR Counter= 5 downto 1 do
        PRINT Counter
```

EXAMPLES

(1)

```
X=10
WHILE X > 0 DO
BEGIN
        PRINT X
        X= X- 3
END
```

(2)

```
X= 10
WHILE X > 10 DO
        X= X- 3
PRINT X
```

(3)

```
X= 1
WHILE X  < 10 DO
BEGIN
        PRINT  X*2
        X= X+1
END
```

GRADE COMPUTATION

	Legend	Frequency	Percentage
Long Test	LT 03 – 01	2	30%
Quizzes	Q 03 – 01	5	25%
Assignment	ASGN 03 - 01	6	20 %
Recitation / Work Ethics			15 %
Problem Set	PS 01	1	10%

Definition

Long Test	- any written long exam to test the learning competencies.
Quiz	- any written exam given by the instructor during the class period.
Assignment	- written home works that are to be passed.
Problem Set	- answers like written pseudocode / programs of the given problem set.
Recitation / Work Ethics	- class participation, attendance and behavior.

REFERENCES

Computer Fundamentals
By: Juny Pilapil Laputt

Introduction to Computer Science
By : Vladimir Zwass

Turbo Pascal : An Introduction to Object-Oriented Programming
By : Larry Joel Goldstein

The C Programming Language (Second Edition)
By : Brian W. Kernighan / Dennis Ritchie

Using Clipper
By: Edward Tiley / QUE